SERIAL PATTERN ACQUISITION: A PRODUCTION SYSTEM APPROACH

D. A. Waterman

Carnegie-Mellon University

Prepared for:

Air Force Office of Scientific Research
Defense Advanced Research Projects Agency
Public Health Service

February 1975

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR - TR - 75 - 1082	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD-A013 569
4. TITLE (and Subtitle) SERIAL PATTERN ACQUISITION: A PRODUCTION SYSTEM APPROACH		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) D. A. Waterman		8. CONTRACT OR GRANT NUMBER(s) F44620--73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Dept. of Computer Science Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101D A0-2466
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE February 1975 13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Air Force Office of Scientific Research (NM) 1400 Wilson Blvd Arlington, VA 22209		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A production system technique for recognizing regularities in serial patterns is presented in the context of the letter series extrapolation problem. The learning technique consists of creating an ordered set of production rules to represent the concept of a pattern, such that each rule is a hypothesis about which pattern contexts lead to which new pattern elements. The production system learning technique is compared with other series extrapolation methods and examples of series concepts learned by a computer implementation of the technique are given.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SERIAL PATTERN ACQUISITION:

A PRODUCTION SYSTEM APPROACH

by D. A. Waterman

Department of Psychology
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Complex Information Processing

Working Paper #286

February, 1975

This work was supported in part by the Defense Advanced Research Projects Agency under contract F44620-73-C-0074 monitored by the Air Force Office of Scientific Research and in part by the NIH Grant MH-07722.

ABSTRACT

A production system technique for recognizing regularities in serial patterns is presented in the context of the letter series extrapolation problem. The learning technique consists of creating an ordered set of production rules to represent the concept of a pattern, such that each rule is a hypothesis about which pattern contexts lead to which new pattern elements. The production system learning technique is compared with other series extrapolation methods and examples of series concepts learned by a computer implementation of the technique are given.

SERIAL PATTERN ACQUISITION: A PRODUCTION SYSTEM APPROACH

by D. A. Waterman

A major hurdle to be faced in implementing computer models of complex learning is the basic task of recognizing regularities in data. This is particularly critical for so called "induction" type learning where a large number of specific data-representations must be mapped into a single more general data-representation. Much work has already been done on induction programs, particularly in the area of pattern recognition (Selfridge and Neisser, 1963; Zobrist, 1971; Uhr, 1973) and sequence extrapolation (Feldman, 1963; Simon and Kotovsky, 1963; Uhr, 1964; Solomonoff, 1964; Ernst and Newell, 1969; Klahr and Wallace, 1970; Williams, 1972; Hedrick, 1974; Hunt and Poltrock, 1974). A somewhat different approach to the problem of machine induction will now be presented.

Ideally, what is needed is a simple uniform technique for recognizing regularities in data, a technique which can be considered a natural extension of basic associative learning techniques such as rote learning. Such a technique would tend to bridge the gap between simple learning like memorizing the addition table, and complex learning like inducing the concept of a series.

In this paper a technique for recognizing regularities will be presented in the context of the series extrapolation problem. No attempt will be made here to generalize this technique to other induction type problems, although some sort of generalization seems feasible. First the problem of data representation will be discussed. Then, the learning technique will be described as it applies to letter series extrapolation problems. Finally, examples will be presented of series concepts learned by a computer implementation of the learning technique.

II. DATA REPRESENTATION

Basic associative learning can be thought of as associating a stimulus A with a response B. This can be represented very naturally as a set of production rules (Newell and Simon, 1972), since a production rule is just a set of conditions associated with a particular set of actions. Thus a portion of the addition table for integers could be represented as the following ordered set of rules:

$$1,1 \rightarrow 2$$

$$1,2 \rightarrow 3$$

$$1,3 \rightarrow 4$$

$$1,4 \rightarrow 5 \quad .$$

This is interpreted: if you have 1 + 1 then the sum is 2, else if you have 1 + 2 it's 3, etc. Only ordered production systems will be considered, that is, to obtain a result the conditions in the left-hand sides of the rules are compared to elements in some data base, and the highest priority rule (topmost rule) whose conditions all match data base elements has its actions executed.

More complex information, such as letter series concepts can also be expressed in production system notation. For example, the concept of the series CDCDCD can be represented as:

$$\begin{aligned} &1.1 \quad C \rightarrow D \\ &1.2 \quad D \rightarrow C \quad . \end{aligned} \qquad (1)$$

This can be interpreted: if the last letter in the series is C then the next is D, else if the last is D then the next is C. It is clear that this is all that is needed to extend the series indefinitely.

Simple Letter Series Concepts

The concept of a series will be defined to be a set of extrapolation rules, as in (1) above, together with a set of initialization rules. The

extrapolation rules contain enough information to extend the series, but both extrapolation and initialization rules are needed to generate the series from scratch. Initialization in (1) can be provided by including * → C as the last rule of the production system, where the asterisk (*) represents a condition defined to match any data base, even an empty one. Thus if no extrapolation rules match the data base then the initialization rule * → C will match by definition. In this paper the extrapolation rules will be referred to as the concept of the series, with the understanding that the actual concept also includes initialization rules.

Consider the more interesting series, GBDGBGBDGBG. This series is composed of repeated occurrences of the string GBDGB. Furthermore, its description does not require the use of predecessor or successor relations on an alphabet. Series like this which can be described using nothing more than the equality or same relation will be called simple repetition type series. A production system (PS) representation of the concept of this series is shown below.

$$\begin{array}{ll} 2.1 & D\ G\ B \rightarrow G \\ 2.2 & G\ B \rightarrow D \\ 2.3 & D \rightarrow G \\ 2.4 & G \rightarrow B \end{array} \qquad (2)$$

This is interpreted: if the last 3 letters in the series are DGB the next letter is G, otherwise if the last 2 are GB the next is D, etc. The rules are always applied to the growing end of the series and always result in the prediction of a single letter. To indicate that the series starts with G, the initialization rule * → G is needed at the bottom of the production system.

In production system (1) the regularities represented are the facts that

D always follows C, and C always follows D. In production system (2) they are that G always follows the string DGB, D follows all GB's not immediately preceeded by D, G always follows D, and B always follows G. This shows, at least, that a production system representation is adequate for expressing the concept of a simple repetition type series in terms of its regularities.

Sequence Prediction Tasks

In the literature on induction and learning the work most closely related to production system representation of regularities is the analysis made by Restle (1967) of subjects performing sequence prediction tasks. The subjects were given a series of binary events equivalent to a sequence of 1's and 0's, and were asked to predict each event in the sequence, given the partial sequence prior to that event. Pretraining and test sequences were analyzed in terms of generative rules*, i.e., grammar-like replacement rules that could be used to generate the sequences. The test sequence used was 111001000111001..., which has a period size of nine. Figure 1 compares Restle's replacement rules for the test sequence with a production system representation of that sequence. The replacement rules in no sense constitute a production system or even a Markov normal algorithm (Markov, 1954; Galler and Perlis, 1970) for generating the series. Instead they define a grammar which can generate a number of series, including the test sequence. For example, the top replacement rule generates the seventh item of the sequence and is interpreted "if you have 1 then replace it with 0". Thus the test sequence can be generated by starting with 000 and applying the rules as shown below:

$$000 \overset{1}{\Rightarrow} \underline{1} \overset{2}{\Rightarrow} 1\underline{1} \overset{3}{\Rightarrow} 11\underline{1} \overset{4}{\Rightarrow} \underline{0} \overset{5}{\Rightarrow} 0\underline{0} \overset{6}{\Rightarrow} \underline{1} \overset{7}{\Rightarrow} \underline{0} \overset{8}{\Rightarrow} 0\underline{0} \overset{9}{\Rightarrow} 00\underline{0}.$$

*These rules were inferred by a manual analysis of the test sequence, rather than by a computer model of the induction task.

Item Predicted	Replacement Rules	Production Rules
(7)	1 → 0	1 0 0 1 → 0
(6)	0 0 → 1	1 1 0 0 → 1
(3)	1 1 → 1 1 1	0 1 1 → 1
(2)	1 → 1 1	0 0 1 → 1
(9)	0 0 → 0 0 0	1 0 0 → 0
(1)	0 0 0 → 1	0 0 → 1
(8,5)	0 → 0 0	0 → 0
(4)	1 1 1 → 0	1 1 → 0

Figure 1. Comparison of Restle's replacement rules and production system rules for the series with period 111001000.

The reason the replacement rules generate series other than the test sequence is that some rules (7 and 2, 6 and 9) contain identical left hand sides. Restle found that subjects make the most errors predicting items that these "optional" rules generate.

The production rules in Figure 1, unlike the replacement rules, represent the concept of the test sequence since they have associated with them a general control mechanism (interpreter for ordered PS's) which defines their use. Notice, however, that the replacement and production rules are pair-wise isomorphic, i.e., for each replacement rule that predicts a symbol there is a corresponding production rule that predicts the same symbol. The production rules which correspond to the "optional" replacement rules are the most complex, since they have the most symbols in their left hand sides. This occurs because enough context must be retained in the left hand side of the production rule to discriminate between similar alternatives. Thus within the PS framework one would expect the most errors during learning to occur on items generated by the most complex rules, which corresponds to the result obtained by Restle. We will now consider the problem of generating a concept from a series and will describe a learning technique capable of creating the production rule representation shown in Figure 1.

III. BASIC LEARNING TECHNIQUE

A learning technique will now be described that is a simple, uniform procedure for generating the concept of a series by finding regularities in the series. In general terms, the technique consists of creating a hypothesis about a particular type of regularity in the data, adding this hypothesis, in the form of a production rule, to the current set of hypotheses (the production system), and then using the data to test the hypotheses. When the data prove a hypothesis false, a new hypothesis is added above the

error-causing one.

In terms of series concepts, each hypothesis consists of a production rule formed from a consecutive sequence of letters from the series (the condition) and the letter assumed to follow that sequence (the action). The action always consists of just a single letter.* Sequences of the series are presented to the production system (first letter, first-two letters, first-three letters, etc.) and it predicts what the next letter should be. The prediction is checked by comparing it to the next letter in the actual series. When the prediction is in error a new rule is added to the system above the error-causing rule. The new rule contains one more letter in its condition than the error-causing rule and the actual next letter as its action. The principle is one of minimum local consistency. A new rule is always a correct statement about the sequence, and is only created following an error at precisely that point in the sequence. When no prediction is made (the sequence of letters fails to match any of the rules) a new rule with a condition equal to the rightmost letter of the sequence and an action equal to the actual next letter is added.

A learning cycle for a series containing n+1 letters consists of presenting the system with the first letter, the first-two letters, on up through the first-n letters, and obtaining a prediction in each case. The learning phase consists of repeated learning cycles and is complete when a learning cycle is encountered which produces nothing but correct predictions. At this point the production system represents the concept of the series and can be used to predict the extensions of the series.

*Rules which predict more than one letter can also be used to form production system concepts of series. Such systems can be generated using the same learning techniques described in this paper. One problem with such systems is that they generate unduly complicated rules when the number of letters predicted by the rules exceeds the period size of the series being represented

An example of this technique applied to the series GBDGBGB will now be presented. Initially the system contains no rules and thus fails to predict the first letter of the series. This error leads to the creation of the default rule * → G. Now G is given to the system and matches the default rule. This is considered an error* so the rule G → B is added. Now GB is presented, does not match G → B, but does match the default rule. Since this is considered an error B → D is added. Next GBD is presented and also matches only the default rule, leading to the addition of the rule D → G. The system now looks like:

$$\begin{aligned} 3.1 \quad & D \rightarrow G \\ 3.2 \quad & B \rightarrow D \\ 3.3 \quad & G \rightarrow B \\ 3.4 \quad & * \rightarrow G \quad . \end{aligned} \tag{3}$$

Next GBDG is presented, which matches 3.3, predicting that the next letter is B. From the series we see this is indeed the next letter so no new rule is necessary. Next, GBDGB is presented which matches 3.2, predicting that the next letter is D. From the series we see the next letter is actually G, so the rule G B → G is added. Next GBDGBG is presented and matches 3.3, correctly predicting B. Now the first cycle is complete, but since errors occurred the process starts over, and G is presented to the system, which is now:

$$\begin{aligned} 4.1 \quad & G\ B \rightarrow G \\ 4.2 \quad & D \rightarrow G \\ 4.3 \quad & B \rightarrow D \\ 4.4 \quad & G \rightarrow B \\ 4.5 \quad & * \rightarrow G \quad . \end{aligned} \tag{4}$$

*Default (or initialization) rules are always considered to make erroneous predictions in order to accelerate the learning process.

G matches 4.4 and the correct prediction is made. But now GB is presented and leads to an incorrect prediction, thus G B → D is added. GBD and GBDG both elicit correct predictions but GBDGB matches G B → D which predicts D instead of G. Thus D G B → G is added. After one more correct prediction the third cycle begins, but this time all predictions are correct and thus the learning phase terminates. Figure 2 diagrams the rule acquisition process for this particular series, showing the first two cycles. The rules learned are:

$$
\begin{array}{ll}
5.1 & D\ G\ B \rightarrow G \\
5.2 & G\ B \rightarrow D \\
5.3 & G\ B \rightarrow G \\
5.4 & D \rightarrow G \\
5.5 & B \rightarrow D \\
5.6 & G \rightarrow B \\
5.7 & * \rightarrow G
\end{array}
\qquad (5)
$$

We will consider the concept of the series to be the set of non-redundant* rules learned, i.e., the rules that can be accessed using this series as context. We see that G B → D (rule 5.2) makes G B → G (rule 5.3) unconditionally redundant, and B → D (rule 5.5) contextually redundant (since, in this particular series, G always occurs before B). The default rule is always contextually redundant. Removing these redundant rules from production system (5) gives the concept of the series as shown in production system (2).**

This learning technique will handle all letter series based on simple repetition and thus is a theory for recognizing regularities in such series.

---

*See Waterman (1970) for a discussion of redundancy as applied to ordered production systems.

**The system does not have to remove these rules since their presence cannot affect system output. In the current implementation the rules are left in; however, they could be removed by having the system keep track of non-firing rules, eventually eliminating them.

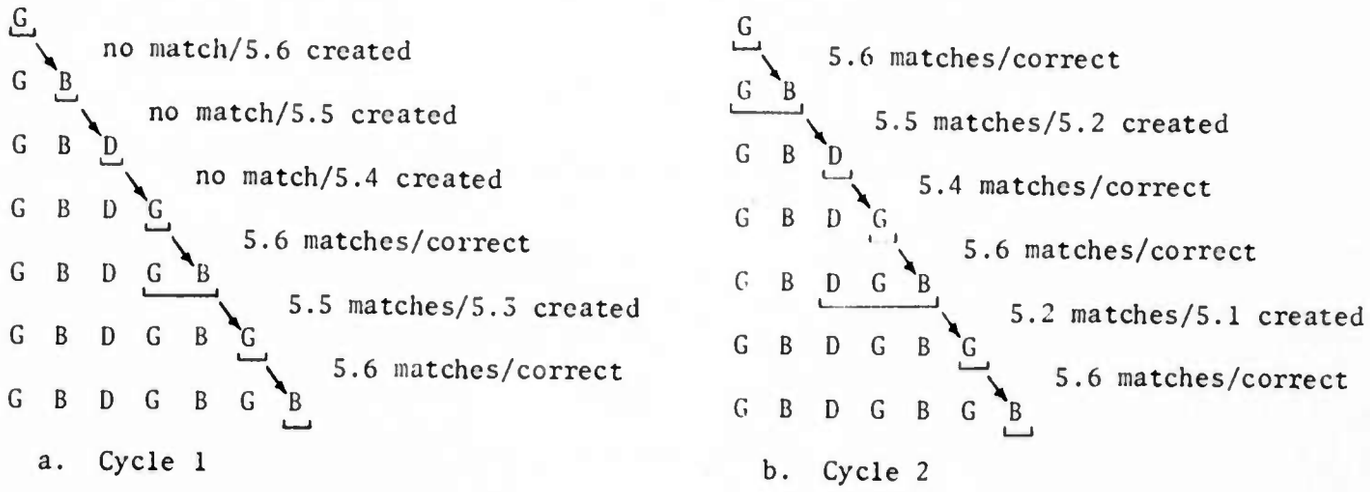


Figure 2. Diagram of Learning Technique on GBDGBGB for first two cycles. (Underlined letters at tail of arrow indicate letters used as rule left hand sides. Letter at head of arrow is rule right hand side).

In fact, it is an instantiation of the compound stimulus hypothesis (Restle and Brown, 1970) in which a response is assumed to be associated with some sequence of adjacent past events. Restle and Brown found a positive but weak relationship between number of errors at a position and number of previous events required to specify the next event. During production system learning the number of errors made at each position in the pattern tends to be proportional to the number of elements in the condition side of the rule that predicts an element for that position. This is true because each error during learning is corrected by effectively adding one new stimulus element to the condition side of the error-causing rule. Now we will see how an extension of this technique can be applied to more complex letter series extrapolation problems.

IV. REPRESENTATION OF COMPLEX LETTER SERIES

The simplest type of letter series other than those characterized by simple repetition are those requiring the use of predecessor and successor operations on the alphabet or any explicitly defined ordered list of symbols. Examples of such series are ABCDEF, AAABBBCCC, and DEFGEFGH. To represent series of this type, the system must be able to handle the concept of variables and must be given the capability for executing both predecessor and successor operations on the alphabet.

Production System Representation

In the production system representation of complex letter series variables will be indicated by the symbols x1, x2, x3, ..., and predecessor and successor operations by an apostrophe (') before or after a variable. Thus 'x1 represents the predecessor of x1, and x1' its successor. A variable in the condition side of a rule matches anything and is temporarily bound to the value of what it matches, thus a bound variable can be used

in the action side of a rule.

With these refinements, the concept of the series ABCDEF can be represented as x1 → x1'. This rule is interpreted: if the last letter of the series is any letter then the next letter in the series is the successor of that letter. Initialization would be accomplished by the rule * → A. Conversely, the series ZYXWVU can be represented as x1 → 'x1, with * → Z for initialization.

Simple repetition can now be represented in a very compact manner, i.e., consider the two simple repetition series discussed earlier. The first, CDCDCD, instead of requiring the two rules shown in production system (1) only requires one rule to represent its concept:

$$x1\ x2 \rightarrow x1. \tag{6}$$

The second series, GBDGBGBDGBG, instead of requiring the four rules shown in production system (2) also requires only one rule to represent its concept:

$$x1\ x2\ x3\ x4\ x5 \rightarrow x1. \tag{7}$$

It should be clear that any simple repetition series of period n can be represented by a single rule of the form:

$$x1\ x2\ x3\ \ldots\ xn \rightarrow x1. \tag{8}$$

Now consider the more complicated series AZCXEVGT. Its concept can be represented as:

$$\begin{array}{l} x1\ x2\ x1'' \rightarrow ''x2 \\ x1\ x2 \rightarrow x1'' \quad , \end{array} \tag{9}$$

where double apostrophes stand for double predecessor or successor. If we apply these rules to the series the first rule fails to match (since T is not the double successor of V) but the second matches, predicting that the next letter is I. If the rules are now applied to AZCXEVGTI, the first

rule matches, predicting the letter R. Thus (9) can be used to extrapolate the series as shown below.

AZCXEVGTIRKPMNO ... (10)

Comparison with other Representations

Other programs have been written which solve letter series extrapolation problems (Simon and Kotovsky, 1963; Klahr and Wallace, 1970; Williams, 1972; Hedrick, 1974; Hunt and Poltrock, 1974). The Klahr and Wallace (K&W) model represents series concepts solely on the basis of inter-period relations, i.e., relations between letters occupying the same relative position in adjacent periods. For example, letting s stand for same, n for next, p for prior, n^2 for double next, and p^2 for double prior, the concept of series (10) would be: n^2p^2. The number of relations is the period size (in this case 2), and the representation is called the pattern template. Simple repetition is represented as a sequence of m same's, where m is the period size. Thus the concept of GBDGBGBDGBG is just sssss.

The Hedrick model represents series concepts as a set of unordered, grammar-like productions which can be used to parse a given input sequence to determine if it is an instance of the series in question. For example, the series ABCDEF... would have a representation equivalent to the grammar:*

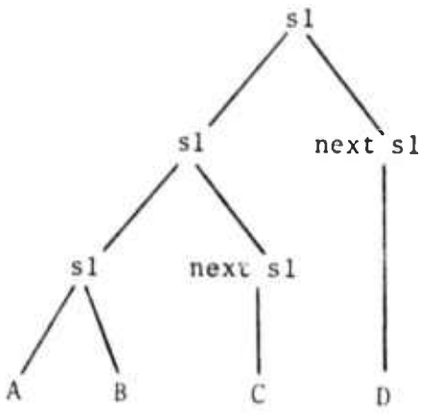
s1 → A B

s1 → s1 next(last letter of s1) .

Thus when given the sequence ABCD the system would recognize it as an instance of s1 (the series ABCDEF...), since the above rules lead to the

*This is a gross simplification of the actual representation. The rules are condition-action pairs where the conditions are pattern matches on both the series and an intermediate semantic net which can be modified by the actions. Thus the model is effectively a production system implementation of a grammar.

parse shown below.



The Hedrick model learns the concept of a series from a set of examples (positive instances) by creating and generalizing productions which classify the components of the series. The model would have to be given AB, ABC, and ABCD before it could acquire the concept of the above series.

The Williams model is part of a more general program for inducing performance strategies from examples taken from aptitude tests. Series concepts are represented in a way very similar to the template representation of Klahr and Wallace. Rules are constructed which define the inter-period relations same and next, one rule for each element in the period. For example, series (10) would be

Rule	Relation	Iteration	Start	Move	Alphabet
1.	next	2	1	2	Forward English
2.	next	2	2	2	Backward English.

Rule 1 states that the double next (next with iteration 2) relation on the forward alphabet holds between letters which are 2 positions apart, starting at position 1. Rule 2 is the same except that the starting point is position 2 and the alphabet is the backward one. This representation is essentially a generalization of the template representation.

The Hunt and Poltrock model represents series concepts on the basis of both inter-period and intra-period relations. This model uses the same three basic relations used by the other models: same, next, and predecessor. These relations can be applied either to adjacent letters within a period or to letters with corresponding positions in adjacent periods. The series concept is represented as a set of rules, one for each letter in the period, and relates each letter to some other letter in the series. A series of period n is shown below.

$$s_1\ s_2\ s_3 \cdots s_n\ z_1\ z_2\ z_3 \cdots z_n$$

The model represents the series as n rules, the first relating z_1 to either s_n or s_1, the second relating z_2 to either z_1 or s_2, etc. Thus the concept of the series AAABBBCCC would be:

$$z_1 = \text{next}(s_3)$$
$$z_2 = \text{same}(z_1)$$
$$z_3 = \text{same}(z_2)\ .$$

Simple repetition is handled by a set of inter-period rules. To illustrate, the concept of GBDGBGBDGBG is shown below.

$$z_1 = \text{same}(s_1)$$
$$z_2 = \text{same}(s_2)$$
$$z_3 = \text{same}(s_3)$$
$$z_4 = \text{same}(s_4)$$
$$z_5 = \text{same}(s_5)$$

Initialization information, such as s_1 = G, s_2 = B, s_3 = D, s_4 = G, and s_5 = B, must also be included as part of the concept.

The Hunt and Poltrock model does not recognize multiple next or predecessor relations; nor does it permit the description of relations between letters with non-corresponding positions in adjacent periods. Thus the concept of series (10) cannot be described. However, this is more a deficiency of the model than of the representational technique used to describe series concepts.

The Simon and Kotovsky (S&K) model represents series concepts primarily on the basis of intra-period relations. This requires a mechanism for stepping a pointer forward through an arbitrary alphabet (the successor operation), a mechanism for resetting the pointer to any arbitrary location in the alphabet, and a mechanism for constructing arbitrary circular alphabets. The standard forward and backward circular alphabets are initially available. The concept of the series AAABBBCCC would be:

$$\begin{array}{l} m1 = [\text{alphabet}];A \\ m1,m1,m1,n(m1) \quad , \end{array} \tag{11}$$

where m1 is the forward alphabet with pointer initialized to A. The n(m1) represents the act of stepping the pointer to the next position in the alphabet and does not represent the generation of a letter of the series, as do the m1's. The concept of series (10) would be:

$$\begin{array}{l} m1 = [\text{alphabet}];A \\ m2 = [\text{backward alphabet}];Z \\ m1,n(m1),n(m1),m2,n(m2),n(m2) \quad . \end{array} \tag{12}$$

Here two separate alphabets, m1 and m2, are required. Simple repetition can be handled by creating an arbitrary alphabet from the letters comprising one period of the series, i.e., the concept of GBDGBGBDGBG would be:

$$\begin{array}{l} m1 = [GBDGB];G \\ m1,n(m1) \quad . \end{array}$$

A comparison of the PS, S&K, and K&W representations is given in Table 1, using series taken from Simon and Kotovsky (1963). Note that in the S&K notation inter-period relations are implicit rather than explicit, while in the K&W notation intra-period relations cannot be described at all. However in production system notation both can be explicitly described, as illustrated by the first two columns of the Table.

Series	PS inter-period	PS intra-period	S&K	K&W
1. ABABAB	x1 x2 → x1 x1 → B * → A	B → A A → B * → A	m1, n(m1) m1 = [AB];A	ss AB
2. AAABBBC	x1 x2 x3 → x1' x1 x2 → A x1 → A * → A	x1 x1 x1 → x1' x1 → x1 * → A	m1, m1, m1, n(m1) m1 = [alphabet];A	nnn AAA
3. ABMCDMEF	x1 M x3 x1'' → M x1 x2 x3 → x1'' x1 x2 → M x1 → B * → A	M x1 x1' M x1'' → x1''' x1 x1' M → x1'' x1 x1' → M C → D A → B * → A	m1, n(m1), m1, n(m1), M m1 = [alphabet];A	n^2n^2s ABM
4. DEFGEFGHFG	x1 x2 x3 x4 → x1' x1 x2 x3 → G x1 x2 → F x1 → E * → D	x1 x1' x1'' x1''' → x1' x1 → x1' * → D	m1, n(m1), m1, n(m1), m1 n(m1), m1, n(m2), E(m1,m2) m1 = m2 = [alphabet];D	nnnn DEFG
5. URTUSTUTT	U x2 x3 U → x2' x1 x2 x3 → x1 x1 x2 → T x1 → R * → U	x1 T U → x1' U x1 T → U U x1 → T U → R * → U	U, m1, n(m1), T m1 = [alphabet];R	sns URT
6. NPAOQAPR	x1 A x3 x1' → A x1 x2 x3 → x1' x1 x2 → A x1 → P * → N	x1 x1'' A → x1' x1 A 'x1 → x1' x1 x1'' → A N → P * → N	m1, n(m1), m2, n(m2), A m1 = [alphabet];N m2 = [alphabet];P	nns NPA

Table 1. Comparison of Production System, Simon & Kotovsky, and Klahr & Wallace notations for representing letter series concepts. (Extrapolation rules are shown above the dotted lines, initialization rules below the dotted lines.)

One advantage of using a PS representation is that it permits initialization rules to be represented in a form identical to extrapolation rules. Furthermore, there is a certain degree of independence between initialization and extrapolation which makes it possible to extrapolate a given series without using initialization information. With the K&W representation this is also possible, but the system must effectively regenerate the series from scratch in order to extrapolate it. To extrapolate a series using S&K extrapolation rules the system must obtain the initialization information from the series (a non-trivial task) and then use it to regenerate the series from scratch.

Representation of Hierarchical Sequences

Sequential behavior can be analyzed in terms of hierarchical systems (Chomsky, 1963; Restle, 1970) and we will now compare one such analysis with a corresponding production system analysis. Restle (1970) developed a notation for describing a hierarchical sequence as a series of nested operators: T_i, R and M, which can transpose (add or subtract by i), repeat, or mirror (reflect) sequences given as arguments. For example, $T_{+1}(3)$ is (3 4) and $T_{+3}(3\ 4)$ is (3 4 6 7). Similarly, R(3) is (3 3) and R(1 2) is (1 2 1 2)*. Thus the pattern 3 1 3 1 6 4 6 4 can be represented as $T_{+3}(R(T_{-2}(3)))$. This is equivalent to representing the pattern as a regular binary tree.

The hierarchical pattern 3 1 3 1 6 4 6 4 can also be represented by the following production system:

$$\begin{aligned} &13.1 \quad x1\ x2\ x3\ x4 \rightarrow x1'''\\ &13.2 \quad\quad\quad x1\ x2 \rightarrow x1 \\ &13.3 \quad\quad\quad\quad\ x1 \rightarrow ''x1 \quad . \end{aligned} \tag{13}$$

*For a description of the "mirror" operation see Restle (1970).

The pattern is generated from the initial element 3 by one application of rule 13.3 (to produce 3 1), two applications of 13.2 (to produce 3 1 3 1) and four applications of 13.1 (to produce 3 1 3 1 6 4 6 4). Note that each production rule is analogous to one of the Restle operators (or one level in the corresponding binary tree). Hierarchical sequences based on transposition and repetition can be described in terms of this PS notation since these operations map directly into the predecessor, successor, and same relations used by the PS's in this paper.

Greeno and Simon (1974) have analyzed the problem of converting sequence information stored as a hierarchy of operators into serially ordered performance. The analysis was made on information represented in Restle's notation and considered questions about the requirements made by the interpretive process on memory storage and computational complexity. Three interpretive processes (push down, recompute, and doubling) were presented for producing the sequence 5 6 2 3 4 5 1 2 from $T_{-1}(T_{-3}(T_1(5)))$, and each was analyzed in terms of storage and computational requirements. One of these processes, called doubling, involves the application of identical operators several times in succession, as illustrated in Figure 3. This particular interpretive process is identical to the one used to interpret a production system representation of this pattern. For example, the above sequence can be represented in PS form as:

$$\begin{aligned} &14.1 \quad x1\ x2\ x3\ x4 \rightarrow {}'x1 \\ &14.2 \quad\quad x1\ x2 \rightarrow {}'''x1 \\ &14.3 \quad\quad\quad x1 \rightarrow x1' \quad . \end{aligned} \tag{14}$$

If we map rule 14.1 into the operator T_{-1}, 14.2 into T_{-3}, and 14.3 into T_1, we see that the sequence of rule applications which generates the series is the same sequence given in Figure 3. Greeno and Simon found that the

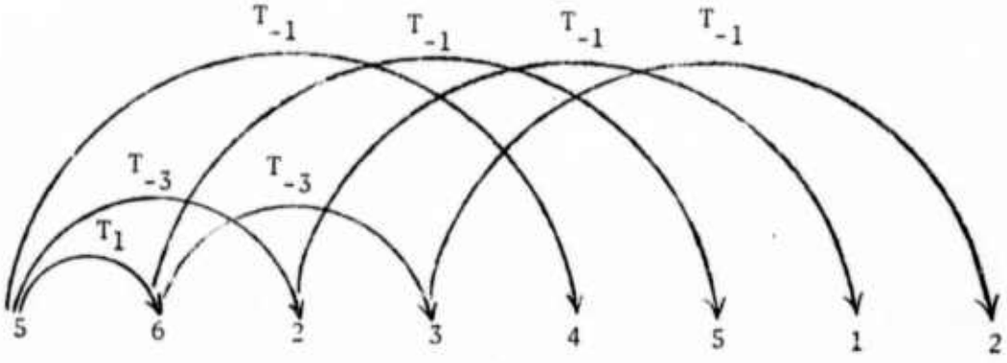


Figure 3. Doubling interpretive process for producing a sequence from $T_{-1}(T_{-3}(T_1(5)))$. (Taken from Greeno and Simon, 1974).

doubling interpretation process, when compared with the other two, minimized the number of operator applications and operator retrievals from memory, while maximizing the amount of short-term memory required.* Thus we conclude that a production system representation of serial patterns implies a process for which computational complexity has been reduced at the expense of memory requirements.

V. EXTENSION OF LEARNING TECHNIQUE

The primary interest here is in developing a simple uniform technique for generating the concept of a complex letter series. The creation of compact or minimal sets of rules is considered to be of secondary importance. An extension of the previously discussed learning technique will now be presented. Since series based on alphabets are used, rules must now be generalized before being added to the system. For example, the series ABCD cannot be extrapolated from the rules A → B, B → C, and C → D. A generalized version of these rules, namely x1 → x1', provides the needed predictive power. But the need to generalize rules leads to another problem: that of determining which relations between letters should be made explicit in the generalization. This is a non-trivial problem because the system will either make errors or become bogged down in backtracking if spurious relations are made explicit (Waterman, 1974). This problem is solved by hypothesizing a period size and then making explicit only relations between letters which occupy the same relative position within adjacent periods. This method for limiting the search for relevant relations is called the template strategy. Since this technique deals only with inter-period relations it creates production systems similar to those shown in column 1 of Table 1.

*For m operators the number of operator applications was $2^m - 1$, operator retrievals was m, and maximum memory capacity 2^{m-1}.

Example of Production System Series Extrapolation

The new production system learning technique is identical to the earlier one with the following exceptions:

1. Only one cycle through the series is necessary, regardless of errors.
2. New rules are added immediately above the error-causing rule, rather than above all the current rules.
3. A generalized version of each rule is added to the system, rather than a specific one, and only inter-period relations are made explicit.
4. Period size is hypothesized, in order from 1 to n, where n is the length of the given series. For each period size hypothesis one learning cycle is attempted. The cycle is aborted and the period size hypothesis incremented whenever (a) no relation can be found between letters occupying the same relative position in adjacent periods, or (b) the number of rules added exceeds the period size hypothesis.

An example using the series ABMCDMEF will illustrate this procedure. The initial period size hypothesis is 1, and no rules are present. The context A is presented to the system; since there are no rules an error results and the rule A → B is generalized and added to the system. Since the period is assumed to b 1 the relation between A and B is made explicit and A → B is added as x1 → x1'. Next the context AB is presented which matches the rule just added and C is predicted, rather than the correct letter M. Thus a new rule must be added. However this would make the number of rules (2) larger than the period size (1), so the cycle is aborted and starts over with a period size hypothesis of 2 and no rules present.

Now the context A is presented; it leads to an error since no rules are present, and the rule A → B is generalized and added as x1 → B, since A and B are now both in the same period. Next AB is presented which matches the rule just added and B is predicted rather than the correct letter M. Thus the rule A B → M is generalized and added, except that here the generalization fails since no relation can be found between A and M (nothing higher than triple predecessor and successors are considered).

As before the cycle is aborted and starts over with no rules present and a period size hypothesis of 3.

Again the context A leads to an error and x1 → B is added to the system. Then AB is presented, leading to the erroneous prediction of B. Thus A B → M is generalized and added as x1 x2 → M, since A, B, and M are all in the same period. The set of rules is now:

$$\begin{array}{lll} 15.1 & x1\ x2 \rightarrow M & \text{(initialization)} \\ 15.2 & x1 \rightarrow B & \text{(initialization)} \end{array} \quad . \tag{15}$$

Here "initialization" indicates that these are intra-period rules needed for initialization of the series but not for extrapolation. To accelerate learning this type of rule is always considered to lead to an erroneous prediction. Next the context ABM is presented, matching 15.1 which predicts M rather than the correct letter, C. So the rule A B M → C is generalized and added above 15.1 to produce:

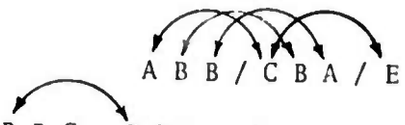
$$\begin{array}{lll} 16.1 & x1\ x2\ x3 \rightarrow x1'' & (1) \\ 16.2 & x1\ x2 \rightarrow M & \text{(initialization)} \\ 16.3 & x1 \rightarrow B & \text{(initialization)} \end{array} \tag{16}$$

where the (1) indicates that this is the first rule added that counts relative to the abort decision based on the number of rules added (only inter-period rules are counted). Next the context ABMC is presented which matches 16.1 and correctly predicts D as the next letter. Now the context ABMCD is presented, again matching 16.1 but incorrectly predicting O. Thus the rule B M C D → M is generalized and added to produce:

$$\begin{array}{lll} 17.1 & x1\ M\ x3\ x1'' \rightarrow M & (2) \\ 17.2 & x1\ x2\ x3 \rightarrow x1'' & (1) \\ 17.3 & x1\ x2 \rightarrow M & \text{(initialization)} \\ 17.4 & x1 \rightarrow B & \text{(initialization)} \end{array} \tag{17}$$

Finally, when ABMCDM and ABMCDME are presented they elicit correct predictions and the learning phase is complete. Now the entire series ABMCDMEF is presented and the correct extrapolation, letter M, is made. The concept of the series is considered to be the extrapolation rules (17.1 and 17.2) plus the initialization rules (17.3 and 17.4) shown in (17).*

Rule generalization is straightforward and requires the rule, the series, and the hypothesized period size. For example, if the rule is B B C → B and the series is ABBCBAE with period size 3, then, as shown below, arrows can be drawn between letters whose relations are to be made explicit.



Now the rule B B C → B has only one such arrow, thus only the relation between the first and last B is made explicit. Since it is a same relation it can be made explicit by using total generalization to get x1 x2 x3 → x1 or partial generalization to get B x2 x3 → B. Partial generalization can only be used for two letters connected by the same relation, and never for letters connected by predecessor or successor relations.

The learning technique just described works when only total generalization on same is permitted and also when only partial generalization on same is permitted. But in the former case the concept learned for series containing simple repetition is much more compact. Thus in the computer implementation of this learning technique, total generalization on same occurs on the first inter-period rule added to the system during each cycle,

*The default rule * → A is also needed to generate the series from scratch. In the current computer implementation of the extended learning technique all initialization rules, except the default one, are learned during the normal execution of the technique. Thus to generate complete series the system must be given either the first letter, the default rule, or a trivial program modification which causes automatic generation of the default rule.

and partial generalization occurs on all the subsequent rules added. Since total generalization on same always leads to a single rule representation of simple repetition series, this procedure is equivalent to the heuristic: "check to see if you have a simple repetition series before proceeding with the more complex series extrapolation methods".

Comparison with Other Series Extrapolation Techniques

The production system learning technique just illustrated is a method for learning series concepts based solely on inter-period relations. In this respect it is similar to the K&W template matching technique for series extrapolation. There are some differences, however, First, for template matching the series must always exhibit two complete periods or the template will not be complete, and no predictions can be made. In the PS technique two periods are not always required since the method automatically hypothesizes that the inter-period relations not yet specified are similar to those already learned. For example, the template technique fails on the series AABBACB, even though this can be extrapolated AABBACBDAEBFA. It can be thought of as the series ABABABA interleaved with ABCDEF. The PS technique* applied to AABBACB produces the series concept:

$$\begin{array}{l} x1\ x2\ x3\ x4\ x1 \rightarrow x2'' \\ x1\ x2\ x3\ x4 \rightarrow x1 \quad , \end{array} \tag{18}$$

from which the correct extrapolation can be made. A second difference between the template technique and the PS technique is that the former always finds a concept based on the shortest period whereas the latter

*For this example only, the technique consists of using only total generalization on same.

may find a concept based on some multiple of the shortest period. Even when this occurs the predictions made by the PS technique are identical to those made by the template technique.

A production system learning technique based on intra-period relations has not yet been developed, but might prove to be a promising area for continued research. One of the major problems with this approach is the difficulty during the learning phase of distinguishing between relevant and spurious intra-period relations. Both the Simon-Kotovsky and Hunt-Poltrock models dealt with this problem with a certain degree of success, thus the heuristics they used should provide useful guidelines for an adaptive production system implementation.

Computer Implementation of Production System Learning Techniques

Both the basic learning technique (applied to simple repetition series) and the extended learning technique (applied to series using circular alphabets) have been realized as computer programs written in the PAS-II system (Waterman and Newell, 1973). Each program is a short production system which can modify itself by adding new production rules. The rules added by the system represent the concept of the series being learned. A complete description of these self-modifying production systems is given elsewhere (Waterman, 1974).

Examples of series concepts learned by SC1, the program employing the basic learning technique, are shown in Figure 4. Both Figure 5* and Table 1 (first column) contain series concepts learned by SC2, the program employing the extended learning technique. Here redundant rules have been

*The series in Figure 5 were taken from Williams (1972).

Series	Concept	Predictions
1. AABAABA	B → A A A → B A → A	AAB
2. ABACABA	C → A B A → C B → A A → B	CAB
3. GBDGBGBDG	D G B → G G B → D D → G G → B	BGB
4. BBABCBBBBA	B B B B → A B B B → B C B B → B C → B A B → C A → B B → B	BCB

Figure 4. Series Concepts Learned by the SC1 Series Extrapolation Program.

	Series	Concept	Prediction
1.	CDCDCD	x1 x2 → x1	CDC
2.	AAABBB	x1 x2 x3 → x1'	CCC
3.	ATBATAATB	x1 x2 x3 x4 x5 x6 → x1	ATA
4.	RSRTRURV	R x2 R → x2' x1 x2 → x1	RWR
5.	ABMCDMEF	x1 M x3 x1'' → M x1 x2 x3 → x1''	MGH
6.	DEFGEFGH	x1 x2 x3 x4 → x1'	FGH
7.	QXAPXBQXA	x1 x2 x3 x4 x5 x6 → x1	PXB
8.	ABCDABCEA	C x2 x3 x4 C → x2' x1 x2 x3 x4 → x1	BCF
9.	MABMBCMCDM	M x2 x3 x4 x5 x6 M → x2'' x1 x2 M x4 x5 x6 x1'' x2'' → M x1 x2 x3 x4 x5 x6 x1'' → x2'' x1 x2 x3 x4 x5 x6 → x1	DEM
10.	URTUSTU	U x2 x3 U → x2' x1 x2 x3 → x1	TTU
11.	MNLNKNJ	x1 N 'x1 → N x1 x2 → 'x1	NIN
12.	ABYABXAB	B x2 x3 B → 'x2 x1 x2 x3 → x1	WAB
13.	RSCDSTDE	x1 x2 x3 x4 → x1'	TUE
14.	NPAOQAPR	x1 A x3 x1' → A x1 x2 x3 → x1'	AQS
15.	MNOMOOMPO	M x2 x3 M → x2' x1 x2 x3 → x1	MQO
16.	WXAXYBY	x1 x2 x3 → x1'	ZCZ
17.	JKQRKLRS	x1 x2 x3 x4 → x1'	LMS
18.	PONONMNM	x1 x2 x3 → 'x1	LML
19.	CEGEDEHEEE	x1 E x3 x4 x1' → E x1 x2 x3 x4 → x1'	IEF

Figure 5. Series Concepts learned by the SC2 Series Extrapolation Program.

eliminated from the concept descriptions. Figure 6 contains more difficult series concepts learned by SC2. The S&K program, or any program based on intra-period relations, would tend to have difficulty with these series. Note that series 4 in Figure 6 is another that the K&W template matching procedure would be unable to solve.

Even though the SC2 program is an extension of SC1 they do not always make the same predictions, particularly when given ambiguous series. For example, given the series ABBA, SC1 makes the simple extrapolation: BBA..., or simple repetition of period 3. However, SC2 (and the K&W program) would find a more complex extrapolation of period 2, i.e., CZD... . For unambiguous simple repetition series, SC1 and SC2 always make the same predictions, but SC2 produces a simpler concept after a much greater computational effort.

VI. CONCLUSION

A learning technique has been presented for finding regularities in sequential patterns. It consists of nothing more complex than forming an ordered set of hypotheses about which pattern contexts lead to which new pattern elements. The learning system starts with very general hypotheses, i.e., rules which apply to particular classes of patterns. As these rules are proven erroneous their generality is reduced by adding new rules above them which apply to subclasses of these patterns. More specifically, learning proceeds by first assuming that only one element in a particular pattern context is relevant and then, as this is proven false, falling back to the less general assumption that other elements in that pattern context are also relevant. When the learning phase is complete, the system has learned which pattern elements are relevant given any particular pattern context.

	Series	Concept	Predictions
1.	ABCCDEFFG	x1 x2 x3 x4 → x1'''	HII
2.	DDCCDDBDEAD	x1 D x3 'x1 → D D x2 x3 D → x2' x1 x2 x3 → 'x1	FZD
3.	BADBADCE	x1 x2 x3 x4 'x1 → x2''' x1 x2 x3 x4 → 'x1	ZGB
4.	AAABBBCACBD	x1 A x3 x4 x5 x6 x1'' → A x1 B x3 x4 x5 x6 x1'' → B x1 x2 x3 x4 x5 x6 → x1''	BEA
5.	ABCBCDCDEF	C x2 x3 x4 C x2'' → x3'' x1 C x3 x4 x1'' → C x1 x2 x3 x4 → x1''	CFG
6.	ADUACUAEUABUAF	x1 U A x4 x5 x6 'x1 U A → x4' U A x3 x4 x5 x6 U A → 'x3 A x2 x3 x4 x5 x6 A → x2' x1 U x3 x4 x5 x6 'x1 → U x1 x2 x3 x4 x5 x6 → x1	UAA

Figure 6. Difficult Series Concepts Learned
by the SC2 Series Exatrapolation Program

The preceeding remarks apply to both the basic learning technique and to the extension of that technique. However in the extension of the basic learning technique an additional generalization process is present. This is the process of characterizing the relations between elements of the pattern in a very general way before adding the rule containing these pattern elements to the system. Here a specific rule is made more general subject to constraints imposed by the current strategy for recognizing relations. Only one such strategy (the template strategy) was presented in this paper. As mentioned earlier it involved hypothesizing a period size and only recognizing relations between corresponding elements of adjacent periods. However by making the strategy for recognizing relations a little more sophisticated it should be possible to create a single unified program which can learn series concepts based on both inter-period and intra-period relations.

The production system learning technique was presented primarily as an artificial intelligence implementation of sequence extrapolation, rather than as a model of human problem solving. In fact, comparison with Restle's work indicates that the learning technique may more closely model human sequence prediction than sequence extrapolation, even though the two are very closely related. Since both extrapolation and prediction require pattern acquisition, it might be useful to examine the implications of this learning technique for a general theory of human serial pattern acquisition. First, it implies that some portion of human long term memory is organized in the form of a production system or set of condition-action rules. Second, it implies that these rules have an order imposed on them, i.e., given any rule, one can find the next rule in the list. However,

this ordering does not imply that conditions on rules are accessed serially. The matching of production rule conditions against data in short term memory is considered to proceed in parallel*, leading to a set of rules whose conditions match the data. A single rule is chosen from this conflict set on the basis of relative location in long term memory, and its actions are executed. Thus response latency is not necessarily proportional to production system size. Third, the learning technique implies a memory for the locations of rules recently fired. This follows from the necessity of incorporating new rules into the system in front of error-causing rules. Finally, it implies that learning serial patterns involves a liberal use of memory capacity in order to reduce computational complexity.

It is felt that this learning technique can be generalized to other induction-type tasks. A similar, though much simpler, technique has already been used in a production rule simulation of verbal learning (Waterman, 1974). Another similar, but more complex, technique has been used in a production system program which learns heuristics for draw poker (Waterman, 1970).

*This is currently implemented as a simple serial process.

ACKNOWLEDGMENTS

The author wishes to thank David Klahr for his many ideas relevant to this paper. The comments and criticisms of Herbert Simon, Dick Hayes, and Allen Newell are also gratefully acknowledged. This work has been supported in part by the NIH Grant MH-07722, and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (1-58200-8130).

REFERENCES

Chomsky, N. Formal properties of grammars. In Handbook of Mathematical Psychology. Luce, R. P., Bush, R. R., & Galenter, E. (Eds.), Vol. 2, Wiley, 1963.

Ernst, G. W., and Newell, A. GPS: A Case Study in Generality and Problem Solving. Academic Press, 1969, pp. 232-246.

Feldman, J. Simulation of behavior in the binary choice experiment. Computers & Thought. Feigenbaum, E. A., & Feldman, J. (Eds.), 1963.

Galler, B., & Perlis, A. A View of Programming Languages. Addison-Wesley, 1970.

Greeno, J. G., and Simon, H. A. Processes for sequence production. Psychological Review, Vol. 81, n. 3, 19 '4, pp. 187-198.

Hedrick, C. L. A computer program to learn production systems using a semantic net. Ph.D. Dissertation, GSIA, Carnegie-Mellon University, 1974.

Hunt, E. B., & Poltrock, S. E. The mechanics of thought. In Kantowitz, B. H. (Ed.), Human Information Processing: Tutorials in Performance and Cognition, L. Erlbaum Associates, N.J., 1974, pp. 277-350.

Klahr, D., and Wallace, J. G. The development of serial completion strategies: An information processing analysis. British Journal of Psychology, Vol. 61, 1970, pp. 243-257.

Markov, A. A. The theory of algorithms (tr. from the Russian by J. J. Shorr-kon), U.S. Dept. of Commerce, Office of Technical Services, OTS 60-51085. (Teoriya Algorifmov, USSR Academy of Sciences, Moscow, 1954).

Newell, A., and Simon, H. Human Problem Solving. Prentice-Hall, 1972.

Restle, F. Grammatical analysis of the prediction of binary events. Journal of Verbal Learning and Verbal Behavior, 6, 1967, pp. 17-25.

Restle, F. Theory of serial pattern learning: Structural trees. Psychological Review, Vol 77, no. 6, 1970, pp. 481-495.

Restle, F., & Brown, E. R. Serial pattern learning. Journal of Experimental Psychology, Vol. 83, no. 1, 1970, pp. 120-125.

Selfridge, O. G., and Neisser, U. Pattern recognition by machine. Computers & Thought. Feigenbaum, E. A., and Feldman, J., (Eds.), McGraw-Hill, 1963, pp. 235-267.

Simon, H. A., and Kotovsky, K. Human acquisition of concepts for sequential patterns. Psychological Review, Vol. 70, no. 6, 1963, pp. 534-546.

Solomonoff, R. J. A formal theory of inductive inference. Part II. Information and Control, 7, 1964, pp. 224-254.

Uhr, L. Pattern Recognition, Learning and Thought. Prentice-Hall, 1973.

Uhr, L. Pattern-string learning programs. Behavioral Science, Vol. 9, no. 3, July 1964, pp. 258-270.

Waterman, D. A. Generalization learning techniques for automating the learning of heuristics. Artificial Intelligence, Vol. 1, nos. 1 and 2, 1970, pp. 121-170.

Waterman, D. A., and Newell, A. PAS-II: An interactive task-free version of an automatic protocol analysis system. Proceedings of the Third International Joint Conference on Artificial Intelligence, 1973, pp. 431-445.

Waterman, D. A. Adaptive production systems, CIP working paper #285, Psychology Department, Carnegie-Mellon University, December, 1974.

Williams, D. S. Computer program organization induced from problem examples. Representation and Meaning, Simon, H. A., and Siklossy, L. (Eds.), Prentice-Hall, 1972, pp. 143-205.

Zobrist, A. L. The organization of extracted features for pattern recognition. Pattern Recognition, Vol. 3, 1971, pp. 23-30.